

UNITED STATES PATENT APPLICATION FOR

COMBINED IN-CIRCUIT EMULATOR AND PROGRAMMER

Inventors:

Craig Nemecek

Steve Roe

Prepared by:

WAGNER, MURABITO & HAO, LLP

Two North Market Street

Third Floor

San Jose, California 95113

(408) 938-9060

1
2
3
4
5 **COMBINED IN-CIRCUIT EMULATOR AND PROGRAMMER**
6
7
8

9 **CROSS REFERENCE TO RELATED DOCUMENTS**

10 This application is a continuation-in-part of U.S. Patent Application Serial No.
11 09/975,105 filed October 10, 2001 to Nemecek entitled " Host to FPGA Interface
12 in an In-Circuit Emulation System", which is hereby incorporated. The application
13 is related to, incorporates by reference and claims priority benefit under 35 U.S.C.
14 §119(e) of provisional patent application serial no. 60/243,708 filed October 26,
15 2000 to Snyder, et al. entitled "Advanced Programmable Microcontroller Device"
16 which is also hereby incorporated herein by reference.
17
18

19 **COPYRIGHT NOTICE**

20 A portion of the disclosure of this patent document contains material which
21 is subject to copyright protection. The copyright owner has no objection to the
22 facsimile reproduction of the patent document or the patent disclosure, as it
23 appears in the Patent and Trademark Office patent file or records, but otherwise
24 reserves all copyright rights whatsoever.

25 **FIELD OF THE INVENTION**

26 This invention relates generally to the field of In-Circuit Emulation. More
27 particularly, this invention relates to use of a pod device in an In-Circuit Emulation
28 system to provide both programming and In-Circuit Emulation functions.
29

BACKGROUND OF THE INVENTION

In-circuit emulation (ICE) has been used by software and hardware developers for a number of years as a development tool to emulate the operation of complex circuit building blocks and permit diagnosis and debugging of hardware and software. Such in-circuit emulation is most commonly used currently to analyze and debug the behavior of complex devices such as microcontrollers and microprocessors that have internal structures that are far too complex to readily model using computer simulation software alone.

FIGURE 1 illustrates an exemplary conventional in-circuit emulation arrangement 100 used to model, analyze and debug the operation of a microcontroller device. In this arrangement, a host computer (e.g., a personal computer) 110 is connected to a debug logic block 120 which is further connected to a special version of the microcontroller device that has been developed specially for use in emulation. Operational instructions are loaded from the host computer 110 through the debug logic 120 to the special version of the microcontroller 130. The debug logic 120 monitors operation of the microcontroller 130 as the instructions are executed. Depending upon the application, this operation may be monitored while the special version of the microcontroller 130 is interconnected with the circuitry that is intended to interface a production version of the microcontroller in the finished product under development. Such interconnection may be via simulation within host computer 110 or as actual circuitry or some combination thereof. As the circuit is stepped through its operation, the debug logic gathers information about the state of various components of the microcontroller 130 during operation and feeds that information back to the host computer 110 for analysis.

During the course of the analysis, various trace information such as time stamps, register values, data memory content, etc. may be logged in the host computer 110 for analysis and debugging by the designer. Additionally, it is generally the case that various break points can be defined by the designer that

1 cause the program to halt execution at various points in the operation to permit
2 detailed analysis. Other debugging tools may also be provided to enable the user
3 to debug the operation of the circuit.

4 In-circuit emulation systems such as 100 have a number of disadvantages
5 and limitations. In earlier systems, the microcontroller 130 might have been simply
6 the production version of the microcontroller itself with no special debugging
7 features. In such systems, the information that can be gathered by the ICE system
8 100 is limited to that which is available at the pinouts of the microcontroller 130 (or
9 which can be extracted from the microcontroller using clever programming or
10 special coding supported by the processor).

11 Enhancements to such early systems provided the microcontroller or other
12 processor with an array of built-in debugging tools that use standard pins on the
13 part and built-in instructions that facilitated in-circuit emulation. In such enhanced
14 processors, the emulation tools are integrated into the part and thus become a
15 design constraint for developing and improving the part. Thus, support for the
16 debugging instruction code and the like can increase the cost and complexity of the
17 circuit.

18 Newer systems often use a so-called "bond-out" microcontroller. A bond-out
19 version of the microcontroller is a version of the production microcontroller that has
20 been designed with special wirebonding pads on the chip that are not normally
21 connected in the production wirebonding. The bond-out version connects these
22 pads to pins on the microcontroller package to permit access to otherwise
23 inaccessible points of the circuit to facilitate debugging. This technique is in
24 common use, but has the disadvantage of imposing significant limitations on the
25 circuit layout to permit space and circuitry associated with the special wirebonding
26 pads. Additionally, it is usually the case that substantial interface circuitry and
27 other special circuitry to facilitate the debugging and bond-out has to be added to
28 the circuit. This increases the complexity, size, power consumption and potentially
29 reduces the yield of the production part. Moreover, development resources are
30 required to lay out the bond-out circuitry and pads and do associated design of

1 such bond-out circuitry. Additionally, instruction code must generally be provided
2 and supported for such an implementation. Such resources may have to be
3 applied with every updated version of the part and may significantly impact speed,
4 cost or flexibility in development of improved versions of the part.

5 A third technique, one that is used in the Pentium™ and Pentium Pro™
6 series of microprocessors available from Intel Corporation, provides a special
7 "probe mode" of operation of the processor. When the processor is placed in this
8 probe mode, a number of internal signals are routed to a "debug port" for use by the
9 in-circuit emulation system. This debug port is used to permit the in-circuit
10 emulation system to communicate with the processors at all times and, when
11 placed in probe mode, to read otherwise inaccessible probe points within the
12 processor. Of course, providing such a probe mode requires significant design
13 resources to design in all such probe and debug functions and associated
14 instruction code support into the standard processor. This, of course, increases
15 development cost, chip complexity and chip size. Moreover, such facilities become
16 a part of the processor design which must be carried through and updated as
17 required as enhancements to the original design are developed.

18 19 SUMMARY OF THE INVENTION

20 The present invention relates generally to providing device programming
21 capabilities in an ICE system. Objects, advantages and features of the invention
22 will become apparent to those skilled in the art upon consideration of the following
23 detailed description of the invention.

24 In one embodiment consistent with the present invention, a combined In-
25 Circuit Emulation system and device programmer has a pod assembly used in an
26 in-circuit emulation system has both a real microcontroller used in the In-Circuit
27 Emulation and debugging process as well as a socket that accommodates a
28 microcontroller to be programmed (a program microcontroller). Programming can
29 be carried out over a single interface that is shared between the microcontroller and
30 the program microcontroller and which is also used to provide communication

1 between the real microcontroller and the In-Circuit Emulation system to carry out
2 emulation functions. In order to assure that the emulation microcontroller does not
3 interfere with the programming process for a microcontroller placed in a
4 programming socket, a special sleep mode is implemented in the emulation
5 microcontroller. This sleep mode is activated by a process that takes place at
6 power on in which the a reset line is released with a specified data line held in a
7 logic high state. This provides the advantage of using a single pod and ICE system
8 to also provide programming functions without need for a separate programmer.

9 A combined in-circuit emulation system and programmer consistent with
10 certain embodiments of the present invention has a pod carrying an emulation
11 microcontroller and a socket for programming another microcontroller. A base
12 station has a virtual microcontroller that operates in lock-step synchronization with
13 the emulation microcontroller during emulation operations. An interface connects
14 the pod to the base station, the interface having a clock signal line, a pair of data
15 signal lines, a reset line and a power line, wherein the reset line is connected to the
16 emulation microcontroller, but is not connected to the socket. The emulation
17 microcontroller can be placed in a sleep mode so that a microcontroller residing in
18 the socket can be programmed by receiving programming information from the
19 base station without the programming being disturbed by actions of the emulation
20 microcontroller.

21 A pod assembly for use with a combined In-Circuit Emulation system and
22 device programmer consistent with certain embodiments of the present invention
23 has a device under test and a programming socket for carrying a device to be
24 programmed. An interface connects data lines to programming inputs of the device
25 to be programmed and to the device under test. The data lines carry programming
26 instructions to the device to be programmed during a programming operation, and
27 the data lines are used for communication with the device under test during
28 emulation operations.

29 A method of programming a programmable device consistent with certain
30 embodiments of the present invention, wherein the device residing in a socket of

1 a pod that carries an emulation device used in emulation operations, includes
2 applying power to the pod; sending a control signal to the pod to place the
3 emulation device into a sleeping state; and programming the programmable device
4 residing in the socket while the emulation device is in the sleeping state.

5
6 The above summaries are intended to illustrate exemplary embodiments of
7 the invention, which will be best understood in conjunction with the detailed
8 description to follow, and are not intended to limit the scope of the appended
9 claims.

10 11 BRIEF DESCRIPTION OF THE DRAWINGS

12 The features of the invention believed to be novel are set forth with
13 particularity in the appended claims. The invention itself however, both as to
14 organization and method of operation, together with objects and advantages
15 thereof, may be best understood by reference to the following detailed description
16 of the invention, which describes certain exemplary embodiments of the invention,
17 taken in conjunction with the accompanying drawings in which:

18 **FIGURE 1** is a block diagram of a conventional In-Circuit Emulation system.

19 **FIGURE 2** is a block diagram of an exemplary In-Circuit Emulation system
20 consistent with certain microcontroller embodiments of the present invention.

21 **FIGURE 3** is an illustration of the operational phases of an In-Circuit
22 Emulation system consistent with an embodiment of the present invention.

23 **FIGURE 4** is an illustration of the operational phases of an In-Circuit
24 Emulation system consistent with an embodiment of the present invention viewed
25 from a virtual microcontroller perspective.

26 **FIGURE 5** is a timing diagram useful in understanding an exemplary data
27 and control phase of operation of certain embodiments of the present invention.

28 **FIGURE 6** is a block diagram isolating the host to FPGA interface consistent
29 with an embodiment of the present invention

1 **FIGURE 7** is a flow chart describing the operation of the host to FPGA
2 interface in an embodiment consistent with the present invention.

3 **FIGURE 8** is a block diagram of a pod assembly consistent with certain
4 embodiments of the present invention.

5 **FIGURE 9** is a flow chart of a process for entering a debug mode consistent
6 with an embodiment of the present invention.

7 **FIGURE 10** is a flow chart of a process for entering a programming mode
8 consistent with an embodiment of the present invention.

9
10 **DETAILED DESCRIPTION OF THE INVENTION**

11 In the following detailed description of the present invention, numerous
12 specific details are set forth in order to provide a thorough understanding of the
13 present invention. However, it will be recognized by one skilled in the art that the
14 present invention may be practiced without these specific details or with
15 equivalents thereof. In other instances, well known methods, procedures,
16 components, and circuits have not been described in detail as not to unnecessarily
17 obscure aspects of the present invention.

18
19 **NOTATION AND NOMENCLATURE**

20 Some portions of the detailed descriptions which follow are presented in
21 terms of procedures, steps, logic blocks, processing, and other symbolic
22 representations of operations on data bits that can be performed on computer
23 memory. These descriptions and representations are the means used by those
24 skilled in the data processing arts to most effectively convey the substance of their
25 work to others skilled in the art. A procedure, computer executed step, logic block,
26 process, etc., is here, and generally, conceived to be a self-consistent sequence
27 of steps or instructions leading to a desired result. The steps are those requiring
28 physical manipulations of physical quantities.

29 Usually, though not necessarily, these quantities take the form of electrical

1 or magnetic signals capable of being stored, transferred, combined, compared, and
2 otherwise manipulated in a computer system. It has proven convenient at times,
3 principally for reasons of common usage, to refer to these signals as bits, values,
4 elements, symbols, characters, terms, numbers, or the like.

5 It should be borne in mind, however, that all of these and similar terms are
6 to be associated with the appropriate physical quantities and are merely convenient
7 labels applied to these quantities. Unless specifically stated otherwise as apparent
8 from the following discussions, it is appreciated that throughout the present
9 invention, discussions utilizing terms such as "processing" or "transferring" or
10 "executing" or "releasing" or "instructing" or "issuing" or "halting" or "clearing" or the
11 like, refer to the action and processes of a computer system, or similar electronic
12 computing device, that manipulates and transforms data represented as physical
13 (electronic) quantities within the computer system's registers and memories into
14 other data similarly represented as physical quantities within the computer system
15 memories or registers or other such information storage, transmission or display
16 devices.

17 18 **COMBINED IN-CIRCUIT EMULATOR AND PROGRAMMER IN ACCORDANCE** 19 **WITH THE INVENTION**

20 While this invention is susceptible of embodiment in many different forms,
21 there is shown in the drawings and will herein be described in detail specific
22 embodiments, with the understanding that the present disclosure is to be
23 considered as an example of the principles of the invention and not intended to limit
24 the invention to the specific embodiments shown and described. In the description
25 below, like reference numerals are used to describe the same, similar or
26 corresponding parts in the several views of the drawings.

27 A commercial ICE system utilizing the present invention is available from
28 Cypress Micro Systems, Inc., for the CY8C25xxx/26xxx series of microcontrollers.
29 Detailed information regarding this commercial product is available from Cypress

1 Micro Systems, Inc., 22027 17th Avenue SE, Suite 201, Bothell, WA 98021 Bothell,
2 WA in the form of version 1.11 of "PSoC Designer: Integrated Development
3 Environment User Guide", which is hereby incorporated by reference. While the
4 present invention is described in terms of an ICE system for the above exemplary
5 microcontroller device, the invention is equally applicable to other complex circuitry
6 including microprocessors and other circuitry that is suitable for analysis and
7 debugging using in-circuit emulation. Moreover, the invention is not limited to the
8 exact implementation details of the exemplary embodiment used herein for
9 illustrative purposes.

10 Referring now to **FIGURE 2**, an architecture for implementation of an
11 embodiment of an ICE system of the present invention is illustrated as system 200.
12 In system 200, a Host computer 210 (e.g., a personal computer based on a
13 Pentium™ class microprocessor) is interconnected (e.g., using a standard PC
14 interface 214 such as a parallel printer port connection, a universal serial port
15 (USB) connection, etc.) with a base station 218. The host computer 210 generally
16 operates to run an ICE computer program to control the emulation process and
17 further operates in the capacity of a logic analyzer to permit a user to view
18 information provided from the base station 218 for use in analyzing and debugging
19 a system under test or development.

20 The base station 218 is based upon a general purpose programmable
21 hardware device such as a gate array configured to function as a functionally
22 equivalent "virtual microcontroller" 220 (or other device under test (DUT)). This is
23 accomplished using an associated integral memory 222 which stores program
24 instructions, data, trace information and other associated information. Thus, the
25 base station is configured as an emulator of the internal microprocessor portion of
26 the microcontroller 232. In preferred embodiments, a field programmable gate
27 array FPGA (or other programmable logic device) is configured to function as the
28 virtual microcontroller 220. The FPGA and virtual microcontroller 220 will be
29 referred to interchangeably herein. The base station 218 is coupled (e.g., using a
30 four wire interface 226) to a standard production microcontroller 232 mounted in a

1 mounting device referred to as a "pod". The pod, in certain embodiments, provides
2 connections to the microcontroller 232 that permit external probing as well as
3 interconnection with other circuitry as might be used to simulate a system under
4 development.

5 The FPGA of the base station 218 of the current embodiment is designed
6 to emulate the core processor functionality (microprocessor functions, Arithmetic
7 Logic Unit functions and RAM and ROM memory functions) of the Cypress
8 CY8C25xxx/26xxx series microcontrollers. The CY8C25xxx/26xxx series of
9 microcontrollers also incorporates I/O functions and an interrupt controller as well
10 as programmable digital and analog circuitry. This circuitry need not be modeled
11 using the FPGA 220. Instead, the I/O read information, interrupt vectors and other
12 information can be passed to the FPGA 220 from the microcontroller 232 over the
13 interface 226 as will be described later.

14 In order to minimize the need for any special ICE related functions on the
15 microcontroller 232 itself, the FPGA 220 and associated circuitry of the base station
16 218 are designed to operate functionally in a manner identically to that of
17 microprocessor portion of the production microcontroller, but to provide for access
18 to extensive debug tools including readout of registers and memory locations to
19 facilitate traces and other debugging operations.

20 The base station 218's virtual microcontroller 220 operates to execute the
21 code programmed into the microcontroller 232 in lock-step operation with the
22 microcontroller 232. Thus, the actual microcontroller 232 is freed of any need to
23 provide significant special facilities for ICE, since any such facilities can be
24 provided in the virtual microcontroller 220. The base station 218's virtual
25 microcontroller 220 and microcontroller 232 operate together such that I/O reads
26 and interrupts are fully supported in real time. The combination of real and virtual
27 microcontroller behave just as the microcontroller 232 would alone under normal
28 operating conditions. I/O reads and interrupt vectors are transferred from the
29 microcontroller 232 to the base station 218 as will be described later. Base station
30 218 is then able to provide the host computer 210 with the I/O reads and interrupt

1 vectors as well as an array of information internal to the microcontroller 232 within
2 memory and register locations that are otherwise inaccessible.

3 In the designing of a microcontroller other complex circuit such as the
4 microcontroller 232, it is common to implement the design using the Verilog™
5 language (or other suitable language). Thus, it is common that the full functional
6 design description of the microcontroller is fully available in a software format. The
7 base station 218 of the current embodiment is based upon the commercially
8 available Spartan™ series of FPGAs from Xilinx, Inc., 2100 Logic Drive, San Jose,
9 CA 95124. The Verilog™ description can be used as the input to the FPGA design
10 and synthesis tools available from the FPGA manufacturer to realize the virtual
11 microcontroller 220 (generally after timing adjustments and other debugging).
12 Thus, design and realization of the FPGA implementation of an emulator for the
13 microcontroller (virtual microcontroller) or other device can be readily achieved by
14 use of the Verilog™ description along with circuitry to provide interfacing to the base
15 station and the device under test (DUT).

16 In the embodiment described in connection with **FIGURE 2**, the actual
17 production microcontroller 232 carries out its normal functions in the intended
18 application and passes I/O information and other information needed for debugging
19 to the FPGA 220. The virtual microcontroller 220 implemented within the FPGA of
20 base station 218 serves to provide the operator with visibility into the core processor
21 functions that are inaccessible in the production microcontroller 232. Thus, the
22 FPGA 220, by virtue of operating in lock-step operation with the microcontroller 232
23 provides an exact duplicate of internal registers, memory contents, interrupt vectors
24 and other useful debug information. Additionally, memory 222 can be used to store
25 information useful in trace operations that is gathered by the FPGA 220 during
26 execution of the program under test. This architecture, therefore, permits the
27 operator to have visibility into the inner workings of the microcontroller 232 without
28 need to provide special bondouts and expensive circuitry on the microcontroller
29 itself.

30 The base station 218's FPGA based virtual microcontroller 220, operating

1 under control of host computer 210, carries out the core processor functions of
2 microcontroller 232 and thus contains a functionally exact emulated copy of the
3 contents of the registers and memory of the real microcontroller 232. The ICE
4 system starts both microcontrollers (real and virtual) at the same time and keeps
5 them running in synchronization. The real microcontroller 232 sends I/O data to
6 the base station 218 (and in turn to the ICE software operating on the host
7 computer 210 if required) fast enough to keep the real microcontroller 232 and the
8 virtual microcontroller 220 of base station 218 in synchronization. Whenever the
9 system is halted (i.e., when the system is not emulating), other information such
10 as flash memory programming functions, test functions, etc. can be sent over the
11 interface.

12 Because the microcontroller 232 operates in synchronization with the virtual
13 microcontroller 220, less data needs to be sent over the four wire interface than
14 would be required in an ICE system otherwise. The type of data sent over the lines
15 is allowed to change depending on when the data is sent in the execution
16 sequence. In other words, depending on the execution sequence time, the
17 information over the data lines can be commands to the real microcontroller 232
18 or they can be data. Since the clock frequency of the real microcontroller 232 is
19 programmable, it copies its current clock on one of the lines of the four wire
20 interface. Moreover, the lock-step operation of the microcontroller 232 and the
21 virtual microcontroller 220 allows the virtual microcontroller 220 to not require
22 certain resources of the microcontroller 232 such as timers, counters, amplifiers,
23 etc. since they are fully implemented in the microcontroller 232. In addition, the
24 microcontroller 232 (or other DUT) can be debugged in real time without need for
25 extensive debug logic residing on the microcontroller 232, since all registers and
26 memory locations, etc. are available through the virtual microcontroller 220.

27 In the embodiment illustrated, the basic interface used is a four line interface
28 between microcontroller 232 and base station 218. This interface permits use of
29 a standard five wire Category Five patch cable to connect the microcontroller 232
30 and base station 218 in one embodiment, but of course, this is not to be considered

limiting. The four wire interface 226 of the present embodiment can be functionally divided into two functional portions. A data transport portion 242 carries two data lines in the current embodiment. A clock portion 246 carries a debug system clock plus the microcontroller clock signal for the microcontroller 232. Three additional lines are also provided (not shown) for supply, ground and a reset line. But, the data transport portion 242 and the clock portion 246 are of primary interest, since the supply and reset functions can be readily provided in any other suitable manner.

The two portions of the interface are implemented in the current embodiment using four lines as described, however, in other embodiments, these two portions can be implemented with as few as two wires. In the current embodiment, the microcontroller clock signal can be varied by programming (even dynamically during execution of a program). Therefore, it is desirable to have two clock signals - the microcontroller clock to easily track the microcontroller clock timing as well as a system clock that regulates the data transfer and other operations. However, in other embodiments, particularly where a clock frequency is not changed dynamically, a single clock can be used. The single clock can be multiplied or divided as required to implement the required clocking signals.

The present embodiment using an eight bit microcontroller that only reads eight bits at a time on any given I/O read. Thus, the present microcontroller 232 needs only to effect serializing and transferring a maximum of one eight bit I/O read for each instruction cycle. This is easily accommodated using two data lines transferring four bits each over four system clock cycles. However, using a clock which is two times faster, a single line could equally well transfer the data in the same time. Similarly, four lines could be used to transfer the same data in only two clock cycles. In any case, the objective is to transfer the data in a short enough time to permit the virtual microcontroller 220 to process the data and issue any needed response before the next instruction cycle begins. The time required to accomplish this is held at a minimum in the current invention, since the system synchronization eliminates need for any overhead protocol for transmission of the data.

10001568-10101

1 The current embodiment of the invention uses a four line communication
2 interface and method of communicating between the FPGA within base station 218
3 (acting as a "virtual microcontroller" 220 or ICE) and the real microcontroller device
4 under test (microcontroller 232). The four line communication interface is time-
5 dependent so that different information can be transferred at different times over a
6 small number of communication lines. Moreover, since the two processors operate
7 in lockstep, there is no need to provide bus arbitration, framing, or other protocol
8 overhead to effect the communication between the microcontroller 232 and the
9 virtual microcontroller 220. This interface is used for, among other things,
10 transferring of I/O data from the microcontroller 232 to the FPGA 220 (since the
11 FPGA emulates only the core processor functions of the microcontroller in the
12 current embodiment). A first interface line (Data1) is a data line used by the
13 microcontroller 232 to send I/O data to the FPGA based virtual microcontroller 220.
14 This line is also used to notify the FPGA 220 of pending interrupts. This Data1 line
15 is only driven by the real microcontroller 232. A second data line (Data2), which is
16 bidirectional, is used by the microcontroller 232 to send I/O data to the FPGA based
17 virtual microcontroller of base station 218. In addition, the FPGA 220 uses the
18 Data2 line to convey halt requests (i.e., to implement simple or complex
19 breakpoints) to the microcontroller 232.

20 A third interface line is a 24/48 Mhz debug system clock used to drive the
21 virtual microcontroller 220's communication state machines (the logic used within
22 the state controller to communicate with the microcontroller 232). In the current
23 embodiment, this clock always runs at 24 MHz unless the microcontroller 232's
24 internal clock is running at 24 Mhz. In this case the system clock switches to 48
25 Mhz. Of course, these exact clock speeds are not to be considered limiting, but are
26 presented as illustrative of the current exemplary embodiment. The fourth interface
27 line is the internal microcontroller clock from the microcontroller 232.

28 A fifth line can be used to provide a system reset signal to effect the
29 simultaneous startup of both microcontrollers. This fifth line provides a convenient

1000159-10101

1 mechanism to reset the microcontrollers, but in most environments, the
2 simultaneous startup can also be effected in other ways including switching of
3 power. The reset line can be advantageously used as part of a programming
4 operation as will be described later. Sixth and Seventh lines are provided in the
5 current interface to provide power and ground for power supply.

6 The base station 218's virtual microcontroller 220 communicates with the
7 microcontroller 232 via four signal and clock lines forming a part of the four line
8 interface 226 forming a part of a seven wire connection as described below. The
9 interface signals travel over a short (e.g., one foot) of CAT5 network cable. The ICE
10 transmits break commands to the microcontroller 232 via the base station 218,
11 along with register read/write commands when the microcontroller 232 is halted.
12 The microcontroller 232 uses the interface to return register information when
13 halted, and to send I/O read, interrupt vector, and watchdog information while
14 running. The microcontroller 232 also sends a copy of its internal clocks for the
15 ICE. The four lines of the four line interface are the first four entries in the table
16 below. Each of the signals and their purpose is tabulated below in **TABLE 1**:
17
18
19

| Signal Name | Signal Direction with Respect to Base Station 218 | Description |
|---|---|--|
| U_HCLK (Data Clock or HCLOCK) | In | 24/48MHz data clock driven by microcontroller 232. This clock is used to drive the ICE virtual microcontroller communication state machines. This clock always runs at 24MHz, unless the U_CCLK clock is running at 24MHz — then it switches to 48MHz. |
| U_CCLK (microcontroller Clock or CCLOCK) | In | The internal microcontroller 232 CPU clock. |
| U_D1_IRQ (Data1) | In | One of two data lines used by the microcontroller 232 to send I/O data to the ICE. This line is also used to notify the ICE of pending interrupts. This line is only driven by the microcontroller 232 (i.e., unidirectional). |
| U_D0_BRQ (Data2) | In/Out | One of two data lines used by the microcontroller 232 to send I/O data to the ICE. The ICE uses this line to convey halt requests and other information to the microcontroller 232. This line is used for bi-directional communication. |
| ICE_POD_RST (RESET) | Out | Optional active high reset signal to microcontroller 232. |
| ICE_POD_PW_R (POWER) | Out | Optional power supply to microcontroller 232. |
| ICE_POD_GND (GROUND) | Out | Optional ground wire to microcontroller 232. |

TABLE 1

Synchronization between the microcontroller 232 and the virtual microcontroller 220 is achieved by virtue of their virtually identical operation. They are both started simultaneously by a power on or reset signal. They then track

1 each other's operation continuously executing the same instructions using the
2 same clocking signals. The system clock signal and the microcontroller clock
3 signal are shared between the two microcontrollers (real and virtual) so that even
4 if the microprocessor clock is changed during operation, they remain in lock-step.

5 In accordance with certain embodiments of the invention, a mechanism is
6 provided for allowing the FPGA 220 of base station 218 and the microcontroller 232
7 to stop at the same instruction in response to a breakpoint event (a break or halt).
8 The FPGA 220 has the ability monitor the microcontroller states of microcontroller
9 232 for a breakpoint event, due to its lock-step operation with microcontroller 232.
10 In the process of executing an instruction, an internal start of instruction cycle (SOI)
11 signal is generated (by both microcontrollers) that indicates that the device is about
12 to execute a next instruction. If a breakpoint signal (a halt or break signal - the
13 terms "halt" and "break" are used synonymously herein) is generated by the FPGA,
14 the execution of the microcontroller 232 can be stopped at the SOI signal point
15 before the next instruction starts.

16 Although the SOI signal is labeled as a signal indicating the start of an
17 instruction, the SOI signal is used for multiple purposes in the present
18 microcontroller. It is not required that the SOI signal actually indicate a start of
19 instruction for many purposes, merely that there be a convenient time reference on
20 which to base certain actions. For example, any reference signal that always takes
21 place prior to execution of an instruction can be used as a time reference for
22 reading a halt command. Accordingly, any such available or generated reference
23 signal can be used equivalently as a "halt read" signal without departing from the
24 present invention. That notwithstanding, the SOI signal is conveniently used in the
25 current embodiment and will be used as a basis for the explanation that follows, but
26 should not be considered limiting.

27 Logic within the FPGA 220 of base station 218 allows not only for
28 implementation of simple breakpoint events, but also for producing breakpoints as
29 a result of very complex events. By way of example, and not limitation, a
30 breakpoint can be programmed to occur when a program counter reaches 0x0030,

an I/O write is happening and the stack pointer is about to overflow. Other such complex breakpoints can readily be programmed to assist in the process of debugging. Complex breakpoints are allowed, in part, also because the virtual microcontroller 220 has time to carry out complex computations and comparisons after receipt of I/O data transfers from the microcontroller 232 and before the next instruction commences. After the receipt of I/O data from the microcontroller 232, the FPGA 220 of base station 218 has a relatively long amount of computation time to determine if a breakpoint event has occurred or not. In the event a breakpoint has occurred, the microcontroller 232 can be halted and the host processor 210 is informed.

An advantage of this process is that the FPGA 220 and the microcontroller 232 can be stopped at the same time in response to a breakpoint event. Another advantage is that complex and robust breakpoint events are allowed while still maintaining breakpoint synchronization between the two devices. These advantages are achieved with minimal specialized debugging logic (to send I/O data over the interface) and without special bond-out circuitry being required in the microcontroller device under test 232.

Normal operation of the current microcontroller is carried out in a cycle of two distinct stages or phases as illustrated in connection with **FIGURE 3**. The cycle begins with the initial startup or reset of both the microcontroller 232 and the virtual microcontroller 220 at 304. Once both microcontrollers are started in synchronism, the data phase 310 is entered in which serialized data is sent from the microcontroller to the virtual microcontroller. At the start of this phase the internal start of instruction (SOI) signal signifies the beginning of this phase will commence with the next low to high transition of the system clock. In the current embodiment, this data phase lasts four system clock cycles, but this is only intended to be exemplary and not limiting. The SOI signal further indicates that any I/O data read on the previous instruction is now latched into a register and can be serialized and transmitted to the virtual microcontroller. Upon the start of the data phase 310, any such I/O read data (eight bits of data in the current embodiment)

1 is serialized into two four bit nibbles that are transmitted using the Data0 and Data1
2 lines of the current interface data portion 242. One bit is transmitted per data line
3 at the clock rate of the system clock. Thus, all eight bits are transmitted in the four
4 clock cycles of the data transfer phase.

5 At the end of the four clock cycle data transfer phase in the current
6 embodiment, the control phase 318 begins. During this control phase, which in the
7 current embodiment may be as short as two microcontroller clock periods (or as
8 long as about fourteen clock periods, depending upon the number of cycles
9 required to execute an instruction), the microcontroller 232 can send interrupt
10 requests, interrupt data, and watchdog requests. Additionally, the virtual
11 microcontroller 220 can issue halt (break) commands. If a halt command is issued,
12 it is read by the microcontroller at the next SOI signal. Once the control phase
13 ends, the data transfer phase repeats. If there is no data to transfer, data1 and
14 data2 remain idle (e.g., at a logic low state). To simplify the circuitry, I/O bus data
15 are sent across the interface on every instruction, even if it is not a bus transfer.
16 Since the virtual microcontroller 220 is operating in synchronization with
17 microcontroller 232 and executing the same instructions, the emulation system
18 knows that data transferred during non I/O read transfers can be ignored.

19 **FIGURE 4** shows this operational cycle from the perspective of the virtual
20 microcontroller 220. During the data transfer phase 310, the serialized data is
21 received over Data0 and Data1. It should be noted that prior to receipt of this I/O
22 data, the microcontroller 232 has already had access to this data for several clock
23 cycles and has already taken action on the data. However, until receipt of the I/O
24 read data during the data transfer phase 310, the virtual microcontroller 220 has not
25 had access to the data. Thus, upon receipt of the I/O read data during the data
26 phase 310, the virtual microcontroller 220 begins processing the data to catch up
27 with the existing state of microcontroller 232. Moreover, once the I/O data has been
28 read, the host computer 210 or virtual microcontroller 220 may determine that a
29 complex or simple breakpoint has been reached and thus need to issue a break
30 request. Thus, the virtual microcontroller should be able to process the data quickly

1 enough to make such determinations and issue a break request prior to the next
2 SOI. Break requests are read at the internal SOI signal, which also serves as a
3 convenient reference time marker that indicates that I/O data has been read and
4 is available for transmission by the microcontroller 232 to the virtual microcontroller
5 220.

6 By operating in the manner described, any breakpoints can be guaranteed
7 to occur in a manner such that both the virtual microcontroller 220 and the
8 microcontroller 232 halt operation in an identical state. Moreover, although the
9 virtual microcontroller 220 and the microcontroller 232 operate on I/O data obtained
10 at different times, both microcontrollers are in complete synchronization by the time
11 each SOI signal occurs. Thus, the virtual microcontroller 220 and the
12 microcontroller 232 can be said to operate in lock-step with respect to a common
13 time reference of the SOI signal as well as with respect to execution of any
14 particular instruction within a set of instructions being executed by both virtual
15 microcontroller 220 and the microcontroller 232.

16 A transfer of I/O data as just described is illustrated with reference to the
17 timing diagram of **FIGURE 5**. After the microcontroller 232 completes an I/O read
18 instruction, it sends the read data back to the base station 218 to the virtual
19 microcontroller, since the virtual microcontroller 220 of the present embodiment
20 implements only the core processor functions (and not the I/O functions). The ICE
21 system can expect the incoming data stream for an I/O read to commence with the
22 first positive edge of U_HCLK (the debug or system clock) when SOI signal for the
23 following instruction is at a predetermined logic level (e.g., a logic high). Thus, at
24 time T1, the SOI signal makes a transition to a logic high and one system clock
25 cycle later at time T2, the data transfer phase 310 begins. This timing allows the
26 ICE system to get the read data to the emulated accumulator of base station 218
27 before it is needed by the next instruction's execution. Note that the first SOI pulse
28 shown in **FIGURE 5** represents the first SOI following the I/O read instruction (but
29 could be any suitable reference time signal). Transfer of the data from the

1 microcontroller 232 is carried out using the two data lines (data2 and data1, shown
2 as U_D0_BRK and U_D1_IRQ) with each line carrying four bits of an eight bit word.
3 During this data transfer phase 310, an eight bit transfer representing the I/O read
4 data can take place from the microcontroller 232 to the base station 210 in the four
5 clock cycles between T2 and T3. The control phase 318 starts at time T3 and
6 continues until the beginning of the next data transfer phase 310. The SOI signal
7 at T4 indicates that the next data transfer phase is about to start and serves as a
8 reference time to read the data2 line to detect the presence of any halt signal from
9 the virtual microcontroller 220. The current control phase 318 ends at T5 and the
10 next data transfer phase 310 begins.

11 The base station 218 only transmits break (halt) commands to the
12 microcontroller 232 during the control phase. After the microcontroller 232 is halted
13 in response to the break command, the interface can be used to implement
14 memory / register read / write commands. The halt command is read at the SOI
15 signal transition (T1 or T4). The microcontroller 232 uses the interface to return
16 register information when halted, and to send I/O read, interrupt vector and
17 watchdog timer information while running.

18 To summarize, a break is handled as follows: The ICE asserts U_D0_BRQ
19 (break) to stop the microcontroller 232. When the ICE asserts the break, the
20 microcontroller 232 reads it at the SOI transition to high and stops. The ICE assert
21 breaks during the control phase. The microcontroller 232 samples the U_D0_BRQ
22 line at the rising edge of SOI (at T4) to determine if a break is to take place. After
23 halting, the ICE may issue commands over the U_D0_BRQ line to query the status
24 of various registers and memory locations of the virtual microcontroller or carry out
25 other functions.

26 In the case of an interrupt, if an interrupt request is pending for the
27 microcontroller 232, the system asserts U_D1_IRQ as an interrupt request during
28 the control phase of the microcontroller 232. Since the interrupt signal comes to
29 the virtual microcontroller 220 from the microcontroller 232 during the control
30 phase, the virtual microcontroller 220 knows the timing of the interrupt signal going

1 forward. That is, the interrupt signal is the synchronizing event rather than the SOI
2 signal. In case of an interrupt, there is no SOI, because the microcontroller 232
3 performs special interrupt processing including reading the current interrupt vector
4 from the interrupt controller. Since program instructions are not being executed
5 during the interrupt processing, there is no data / control phase. The virtual
6 microcontroller 220 expects the interrupt vector to be passed at a deterministic time
7 across the interface during this special interrupt processing and before execution
8 of instructions proceeds. Since the virtual microcontroller 220 of the current
9 embodiment does not implement an interrupt controller, interrupt vectors are read
10 from the interrupt controller upon receipt of an interrupt request over the interface.
11 The interrupt vector data is passed over the interface using the two data lines as
12 with the I/O read data, following the assertion of an internal microcontroller IVR_N
13 (active low) signal during the control phase. In the current embodiment, an
14 interrupt cycle is approximately 10 clock cycles long. Since the interrupt service
15 cycle is much longer than the time required to transfer the current interrupt vector,
16 the data is easily transferred using the two data lines, with no particular timing
17 issues.

18 If the microcontroller 232 undergoes a watchdog reset, it asserts the IRQ
19 (interrupt) and BRQ (break) lines indefinitely. The ICE detects this condition and
20 further detects that the microcontroller clock has stopped. This is enough to
21 establish that a watchdog reset has occurred. The ICE applies an external reset,
22 and notifies the ICE software in the host computer 210.

23 Referring now to the block diagram of **FIGURE 6**, the interface between the
24 host processor 210 and the base station 218 of a preferred embodiment of the
25 present invention is illustrated. In this embodiment, the connection between the
26 host processor 210 and the FPGA 220 is advantageously provided using a standard
27 IEEE 1284 parallel printer cable 214 with communication carried out using a
28 modification of standard EPP (enhanced parallel port) communication protocol. Of
29 particular interest in this communication interface is the data strobe connection
30 412, the INIT (initialize) connection 416 and the eight data connections (data line

0 through data line 7) 420. These connection are directly connected to the FPGA with the INIT connection connected to the FPGA RESET pin. The data strobe line 412 is connected to the FPGA configuration clock input and the eight data lines 420 are connected to data input pins of the FPGA.

When the software on the host is started, the INIT connection 416 is driven by the host computer 210 to a logic low causing the FPGA to clear its configuration memory 424 and begin receiving configuration data. The configuration data is stored in configuration memory to define the functionality of the FPGA. This configuration data is clocked in eight bits at a time over the data lines 420 using the data strobe signal as a clock signal. That is, an eight bit word is placed on the interface data lines 420 by host processor 210 followed by toggling the data strobe line to clock the data into the FPGA 220. This unidirectional data transfer from the host computer incorporates a set of design parameters that configure the circuitry of the FPGA 220 to function, in part, as a standard IEEE 1284 EPP interface once the FPGA 220 is programmed and functional. This programming configures the FPGA 220 to have an IEEE 1284 EPP interface with the data lines 420 connected to the FPGA as bidirectional data lines, the configuration clock configured to operate as the IEEE 1284 data clock line connected to data strobe 412 and the INIT line 416 continues to drive the FPGA clear and reset function.

Data transfer continues in this manner until the FPGA 220 is fully programmed by virtue of having received the correct amount of data required by the particular FPGA 220 used in base station 218. Thus, each time the host software is initialized, a data transfer to the FPGA 220 occurs to program the FPGA 220 to function in its capacity of a virtual microcontroller (in this embodiment). Once programming ceases, the FPGA 220 "wakes up" as a virtual microcontroller (or whatever device is programmed into the FPGA 220 in general) and begins to function as the virtual microcontroller. At this point, the interface 214 ceases to function as a unidirectional programming interface and begins to function as a bidirectional communication interface using the programmed operation of the FPGA 220 communicating through its programmed IEEE 1248 EPP parallel

1 communication interface.

2 In the virtual microcontroller mode of operation of the FPGA 220,
3 communication is carried out using the eight data lines 420 as bidirectional data
4 lines compliant with IEEE 1284 EPP parallel communication protocol with the data
5 strobe line 412 used as a data clock and the INIT line 416 continuing to act as a
6 clear and reset signal. INIT line 416 can thus be used to reinitialize the
7 programming of the FPGA 220, for example, to revise a design parameter or to
8 simply restart the ICE system. **TABLE 2** below summarizes the significant
9 connections of this interface.

10

11

| Interface Lines | Program Mode Function | Free Running "Awake" Mode Function |
|-----------------------|---|--|
| Data bits 0 through 7 | Unidirectional data into the FPGA | Bidirectional EPP compliant communication |
| Data Strobe | Unidirectional programming clock | EPP Compliant Data Strobe |
| INIT | Low signal indicates clear configuration memory and prepare to receive new configuration data | Low signal indicates clear configuration memory and enter programming mode - prepare to receive new configuration data |

12

13

14

15 **TABLE 2**

16

17 The programming and communication process between the host 210 and
18 the FPGA 220 is described in flow chart 500 of **FIGURE 7** starting at 502. The host
19 software is loaded and initialized at 506, and asserts a logic low on the INIT line
20 416 to signal a reset and clearing of the FPGA 220's configuration memory 424 at
21 510. In response to this signal, the FPGA 220 clears configuration memory 424 at
22 514. The Host computer 210 then begins transferring a new set of configuration
23 parameters to the FPGA 220 at 520 by strobing data into the FPGA's configuration
24 memory 424. This set of configuration parameters configures the FPGA 220 to

1 As previously mentioned, microcontroller 232 is mounted on a pod assembly
2 attached by a bus 226 to base station 218. Advantageously, in certain
3 embodiments, the pod assembly can be used not only to carry microcontroller 232
4 for purposes of In-Circuit Emulation and debugging operations, but also to program
5 another microcontroller. **FIGURE 8** illustrates a pod assembly 610 for carrying out
6 such a programming process. In this embodiment, microcontroller 232 is mounted
7 on the pod assembly and connected to the bus 226 as previously described.
8 Additional connections are provided to microcontroller 232 for reset and power
9 signals as previously described. In addition, a socket 620 is provided for carrying
10 another microcontroller temporarily to provide programming functions for the
11 microcontroller. All of the connections between the base station 218 and the pod
12 can be made using a standard category 5 cable as previously described.

13 In order to carry out programming functions on a microcontroller mounted
14 in socket 620, the operation of microcontroller 232 should be placed in a mode
15 such that it does not disturb the programming process, i.e., a sleep mode. This is
16 accomplished by placing microcontroller 232 in a sleep mode while simultaneously
17 placing the microcontroller 620 in a programming mode. Programming data can
18 then be clocked into the microcontroller in socket 620 from base station 218
19 without the operation being disrupted by the microcontroller 232.

20 In order to accomplish both debug operations and programming operations
21 using pod assembly 610, microcontroller 232 can be selectively placed in either a
22 debug or a sleep mode. **FIGURE 9** illustrates a process 700 starting at 702 for
23 placing microcontroller 232 in a debug mode for running In-Circuit Emulation and
24 debugging operations. This process starts at 702 after which power is turned on
25 to the pod assembly 610 at 706. After power is turned on the In-Circuit Emulation
26 system holds power and reset lines asserted (at a logic high) at 710 until the
27 microcontroller's power stabilizes. After that, the in-circuit emulation system pulls
28 the data0 line to a logic high level at 714 and then releases the reset line while the
29 data0 signal is still at a logic high at 718.

1 This sequence of events tells the microcontroller 232 that it should enter the
2 debug mode at 722. At this point normal In-Circuit Emulation operations and
3 debug operations can be carried out and the microcontroller 232 begins executing
4 its stored operational code.

5 **FIGURE 10** depicts a process 800 used to place the microcontroller 232 in
6 a sleep mode so that it does not disrupt programming operations of the
7 microcontroller in socket 620. Of course, the microcontroller in socket 620 is
8 placed in a programming mode while the microcontroller 232 is in a sleep mode
9 in order to effect programming of the microcontroller in socket 620. Of course,
10 when microcontroller 232 is in the debug mode, there is normally no
11 microcontroller present in socket 620 to disrupt the debug operation.

12 Process 800 starts at 804 after which power is turned on to the pod
13 assembly at 808. The In-Circuit Emulation system holds power and reset asserted
14 at 812 and watches for the data0 line to go to a logic high indicating stable power
15 at 816. At 820 the In-Circuit Emulation system pulls the data0 line to a logic low
16 at 820 and then releases the reset line while holding the data0 line at a logic low
17 at 824. This sequence of events causes the microcontroller 232 to enter the sleep
18 mode.

19 The In-Circuit Emulation system then uses the data1 line as a programming
20 clock line in order to clock data into the microcontroller in socket 620 at 824. Prior
21 to clocking in programming data, a key code is clocked in at 828. This key code
22 is specific to the particular microcontroller device being programmed in order to
23 prevent unauthorized modifications of the program. If the key code does not match
24 at 832, the microcontroller in socket 620 begins running whatever code is stored
25 internally at 840. If the key matches at 832, the microcontroller in socket 620
26 enters a program mode so that it can accept program code. Data representing
27 program code are then clocked into the programmable microcontroller in socket
28 620 at 848 by applying clock signals to the data1 line and data to the data0 line in
29 order to program the socketed microcontroller being programmed. This process
30 proceeds until all of the program lines have been clocked in at 852 after which the

1 microcontroller halts and power is turned off to the pod at 856. The programming
2 process is now completed at 860 and the microcontroller in socket 620 can be
3 removed for use.

4 Thus, an In-Circuit Emulation system pod assembly can be used not only to
5 carry a real microcontroller 232 or other device under test but can double as a
6 device programmer which is normally a separately purchased item that can costs
7 the user several hundred dollars or more.

8 As described above, the microcontroller 232 is placed in a sleep mode when a
9 device in socket 620 is to be programmed. This sleep mode is triggered by the
10 release of reset while the Data0 line is held at a logic low. If reset is released with
11 Data0 at a logic high, the microcontroller 232 boots into the debug mode. In the
12 preferred embodiment, the reset pin on the device in socket 620 is hard wired to
13 a logic low, so that when power is applied to the device in socket 620 and it
14 receives the key code, the device enters the programming mode.

15 With the above-referenced Cypress Microsystems microcontroller family, the
16 microcontroller looks for receipt of a valid key within about 16 milli-seconds from
17 release of reset or valid power in order to enter the programming mode. Otherwise,
18 whatever code is stored in the microcontroller's memory begins executing. Thus,
19 the above sequence of events is carried out rapidly after power-up in order to cause
20 the microcontroller in socket 620 to enter the program mode. To facilitate this and
21 assure that the entire process is carried out within the time specified, the procedure
22 is carried using a hardware state machine incorporated within the base station 218
23 and implemented as a part of the FPGA. However, this is not to be considered
24 limiting.

25 While the present embodiment is implemented using a processor that does
26 not use pipelined instructions, this is not to be considered limiting. As long as
27 adequate time is available to serialize and transmit data over the interface, the
28 present interface and break management techniques could equally well be
29 implemented in a pipelined processor.

electronic storage medium or transmitted over any suitable electronic communication medium. However, those skilled in the art will appreciate that the processes described above can be implemented in any number of variations and in many suitable programming languages without departing from the present invention. For example, the order of certain operations carried out can often be varied, additional operations can be added or operations can be deleted without departing from the invention. Error trapping can be added and/or enhanced and variations can be made in user interface and information presentation without departing from the present invention. Such variations are contemplated and considered equivalent.

While the invention has been described in conjunction with specific embodiments, it is evident that many alternatives, modifications, permutations and variations will become apparent to those skilled in the art in light of the foregoing description. Accordingly, it is intended that the present invention embrace all such alternatives, modifications and variations as fall within the scope of the appended claims.

What is claimed is: